Lab 9 – Graph

Instructions:

- *You are expected to completed Problems 1 - 3 before attending the lab.*
- *Please submit the hardcopy of your program by the end of the lab.*

1. Your tasks

This lab is to implement graph data structure using adjacent matrix. The initial code for this lab provide you a skeleton to implement the following functions. Your tasks are to complete these functions to make them work.

```
const int MAXNUMVERTICES = 100;

class DiGraph {
private:
	int adjTable[MAXNUMVERTICES][MAXNUMVERTICES]; // store the adjcent matrix
	int size; // store the number of vertices in the graph
	int verticesList[MAXNUMVERTICES]; //keep the list of vertices in the ascending
order
	bool isMark[MAXNUMVERTICES]; // used for traversal


protected:
	int findVertexRec(int vertex, int low, int high);
	int findVertex(int vertex);
	void depthFirstTraversalRec(int vertex);
public:
	DiGraph();
	bool insertVertex(int vertex);
	bool deleteVertex(int vertex);
	bool insertEgde(int fromVer, int toVer, int weight = 1);
	bool deleteEgde(int fromVer, int toVer);
	void print();
	void depthFirstTraversal();
};
```

Problem 1.

Complete the `bool DiGraph::insertVertex(int ver)` method in the class `DiGraph`. This method inserts a new vertex `ver` to the graph. If the vertex is already existed return `false`, otherwise it will be inserted to the graph and return `true`. In specific, you need to insert this vertex into the `verticesList` at the correct position. Remember the `verticesList` store the vertices list in the ascending order. In addition, you also need to expand the `adjTable` and increase the `size` of the graph.

```
bool DiGraph::insertVertex(int ver) {
	if (size >= MAXNUMVERTICES) return false;
	if (findVertex(ver) > -1) return false;
	// TODO
}
```

Example:

```
Problem 01
    0 2 5 6 7 9
---------------
0 | 0 0 0 0 0 1
2 | 0 0 0 0 0 0
5 | 0 1 0 0 0 0
6 | 0 0 0 0 0 0
7 | 0 0 0 0 0 1
9 | 0 0 0 0 0 0
```

Problem 2.

Complete the `bool DiGraph::deleteVertex(int ver)` method in the class `DiGraph`**.** This method removes a vertex `ver` from the graph. If the vertex is not existed return `false`, otherwise it will be removed from the graph and return `true`. In specific, you need to remove this vertex from the `verticesList`. Remember the `verticesList` store the vertices list in the ascending order. In addition, you also need to reduce the size of `adjTable` and decrease the `size` of the graph.

```
bool DiGraph::deleteVertex(int ver) {
	int pos = findVertex(ver);
	if (pos == -1) return false;
	// Problem 02 - TODO


}
```

Example:

```
Problem 02
    0 5 6 7 9
--------------
0 | 0 0 0 0 1
5 | 0 0 0 0 0
6 | 0 0 0 0 0
7 | 0 0 0 0 1
9 | 0 0 0 0 0
```

Problem 3.

Complete `void DiGraph::depthFirstTraversal()` function in the class `DiGraph` that will print out the vertices of the graph in depth first order.

```
void DiGraph::depthFirstTraversal() {
	// Problem 03 - TODO
}
```

Example:

```
Problem 03
Depth First Traversal: 0 9 5 6 7
```

Problem 4. (*to be released in the lab*)